

The following security alert was issued by the Information Security Division of the Mississippi Department of ITS and is intended for State government entities. The information may or may not be applicable to the general public and accordingly, the State does not warrant its use for any specific purposes.

DATE ISSUED: June 21, 2011

SUBJECT: Recent News of SQL Injection Attacks on Government Computers

The recent increase in targeted attacks on government and private networks and use of web applications as the primary attack vector has once again underscored the continued prevalence of web application vulnerabilities as a viable means of exploitation which enables the attackers to exfiltrate sensitive data from the target system or attack website visitors with malicious content. The purpose of this bulletin is to focus the attention on the most common web application vulnerability known as SQL injection and how it can be prevented. SQL injection vulnerabilities exist in web applications where malicious user input is passed to the backend database without proper validation.

The best way to protect your web applications from SQL injection is to ensure developers do not allow client-supplied data to modify SQL statement syntax. All user-supplied data must be thoroughly validated before it is sent to a backend database. Examples of input attributes that should be checked are type, length, and format.

There are typically two ways that a developer or administrator can prevent bad user input: either disallow bad characters (black listing) or only allow required characters (white listing). The second option is generally considered to be more secure because it is possible that filters may fail due to new attack methods, different character encoding, and other attack variables. It is preferable to take the time to determine exactly what input is needed and then to reject any incoming request that does not adhere to the requirements.

There are a number of online resources to assist developers in performing input sanitization. Please refer to the references below for links to these secure programming resources.

RECOMMENDATIONS:

All of the following actions should be considered:

- Validate and escape all user provided input before passing it to the backend database.
- Avoid using dynamic SQL whenever possible. Dynamic SQL refers to any situation in which user-supplied input is concatenated with pre-defined SQL. Stored procedures or parameterized SQL can be used as a safer alternative.
- Apply the principle of least-privilege to web applications that interact with your database. It is a good idea to create an account for your web applications that has as few data access rights as possible to limit the scope of damage in the event that a system is compromised.
- Turn off debugging information as it is often used to gather data for subsequent attacks.
- Review server applications for possible SQL injection vulnerabilities, and apply all necessary code revisions and appropriate vendor patches after appropriate testing.
- Consider implementing Web Application Firewall (WAF) technology on the web servers.
- Consider encrypting the sensitive information in the database.

- If you believe you are experiencing a SQL injection attack, please notify MS-ISAC by sending an email to soc@msisac.org and provide sample webserver logs where possible.

REFERENCES:

Open Web Application Security Project (OWASP)

http://www.owasp.org/index.php/SQL_injection

Web Application Security Consortium (WASC)

http://www.webappsec.org/projects/threat/classes/sql_injection.shtml

Microsoft

<http://blogs.iis.net/nazim/archive/2008/04/28/filtering-sql-injection-from-classic-asp.aspx>

CodePlex

<http://www.codeplex.com/IIS6SQLInjection>

Information Week

<http://www.informationweek.com/news/security/attacks/229900111>

InfoSecurity

<http://www.infosecurity-us.com/view/18265/another-comodo-partner-attacked-using-sql-injection>